COMP 330: Relational Databases 1

Chris Jermaine and Kia Teymourian

Rice University

What is a Database?

A collection of data

Plus, a set of programs for managing that data

Back in the Day...

The dominant data model was the network or navigational model (60's and 70's)

Data were a set of records with pointers between them

Much DB code was written in COBOL

Big problem was lack of physical data independence

- Code was written for specific storage model
- Want to change storage? Modify your code
- Want to index your data? Modify your code
- Led to very little flexibility
    - ▷ Your code locked you into a physical database design!

Some People Realized This Was a Problem

By 1970, EF Codd (IBM) was looking at the so-called relational model

- Landmark 1970 paper, "A relational model of data for large shared data banks"
- Led to the 1981 Turing Award
  - Highest honor a computer scientist receives
  - Analogous to a Nobel Prize

Idea: data stored in "relations"

- A relation is a table of tuples or records
- Attributes of a tuple have no sub-structure (are atomic)

No pointers!

Querying in the Relational Model

Querying is done via a "relational calculus"

Declarative

- You give a mathematical description of the tuples you want
- System figures out how to get those for you

Why good?

- Data independence!
- Your code has no data access specs
- So can change physical org, no code re-writes

Relation Schema

All data are stored in tables, or relations

A relation schema consists of:

- A relation name (e.g., LIKES)
- A set of (attribute_name, attribute_type) pairs
    - Each pair is referred to as an "attribute"
    - Or sometimes as a "column"
- Usually denoted using LIKES (DRINKER string, BEER string)
- Or simply LIKES (DRINKER, BEER)

A Relation

A relation schema defines a set of sets

- Specifically, if $T_1, T_2, ..., T_n$ are the n attribute types
- Where each T_i is a set of possible values
  - Ex: string is all finite-length character strings
  - Ex: integer is all numbers from -2^{31} to $2^{31} - 1$
- Then a realization of the schema (aka a "relation") is a subset of
  - $T_1 \times T_2 \times ... \times T_n$
  - where $\times$ is the Cartesian product operator

A Relation (continued)

So for the relation schema LIKES (DRINKER string, BEER string)

A corresponding relation might be

$$\{(\text{“Chris”}, \text{“Taddy Porter”}), (\text{“Kia”}, \text{“Pabst Blue Ribbon”})\}$$

This is also referred to as a “table”

The entries in the relation are referred to as

- “rows”
- “tuples”
- “records”

Keys

In the relational model, given $R(A_1, A2, ..., A_n)$

A set of attributes $K = \{K_1, ..., K_m\}$ is a KEY of R if:

- For any valid realization R' of R...
- For all t_1, t_2 in R'...
- If $t_1[K_1] = t_2[K_1]$ and $t_1[K_2] = t_2[K_2]$ and ... $t_1[K_m] = t_2[K_m]$...
- Then it must be the case that $t_1 = t_2$

Note: every relation schema MUST have a key... why?

What is a key for LIKES (DRINKER, BEER)?

What is a key for STUDENT (NETID, FNAME, LNAME, AGE, COLLEGE)?

Keys (continued)

A relation schema can have many keys

One is typically designated as the PRIMARY KEY

Denoted with an underline

- STUDENT (<u>NETID</u>, FNAME, LNAME, AGE, COLLEGE)

Foreign Keys

The relational model does not have pointers

Why? Two reasons:

- Not nice mathematically
  - ▷ Mathematical elegance key goal in model design
- Implementation difficult
  - ▷ Move an object? All pointers are invalid!
  - ▷ Can have centralized look-up table
  - ▷ But expensive, plus problem still exists

Foreign Keys (continued)

But we still need some notion of between-tuple references

- LIKES (DRINKER, BEER)
- DRINKER (DRINKER, FNAME, LNAME)
- Clearly, LIKES.DRINKER refers to DRINKER.DRINKER

Accomplished via the idea of a FOREIGN KEY

Foreign Keys (continued)

- LIKES (DRINKER, BEER)
- DRINKER (DRINKER, FNAME, LNAME)

Given relation schemas R_1, R_2

- We say a set of attributes K_1 from R_1 is a foreign key to a set of attributes K_2 from R_2 if...
- (1) K_2 is a candidate key for R_2, and...
- (2) For any valid realizations R_1', R_2' of R_1, R_2...
- For each $t_1 \in R_1'$, it MUST be the case that there exists $t_2 \in R_2'$ s.t...
- $t_1[K_1] = t_2[K_1]$ and $t_1[K_2] = t_2[K_2]$ and ... $t_1[K_m] = t_2[K_m]$

Intuitively, what does this mean?

Queries/Computations in the Relational Model

The original query language was the RELATIONAL CALCULUS

- Fully declarative programming language

next was the RELATIONAL ALGEBRA

- Imperative
- Define a set of operations over relations
- A RA program is then a sequence of those operations
- This is the "abstract machine" of RDBs

Today we use SQL

- Heavily influenced by RC
- Has aspects of RA
- Nastier than either of them!

Overview of Relational Calculus

RC is a variant on first order logic

You say: "Give me all tuples t where $P(t)$ holds"

$P(t)$ is a predicate in first order logic

Predicates

First order logic allows predicates

- predicate: function that evals to true/false
- "It's raining on day X" or $Raining(X)$
- "It's cloudy on day X" or $Cloudy(X)$

Can build more complicated preds using logical operations over them

- and
- or
- not
- implies
- iff

Predicates (continued)

Example: $Raining(X) \rightarrow Cloudy(X)$

Evals to true if either:

- It is not raining on day X, or
- It is raining and cloudy on day X

Example: $Raining(X) \wedge Cloudy(X)$

Evals to true if:

- It is raining and cloudy on day X

Note the difference between them!

- $\longrightarrow$ is like a logical "if-then"

First Order Logic

Just predicates and logical ops?

- You've got predicate logic

But when you add quantification

- $\forall$, $\exists$
- You've got first order logic

Universal Quantification

Asserts that a predicate is true all of the time

Example:

- $\forall(X)(Raining(X) \rightarrow Cloudy(X))$
- This is a zero-arg predicate (takes no params)
- Asserts that it only rains when it is cloudy
- Note: idea of universe of discourse is key!

Example:

- $\forall(X)(Friends(X, Y))$
- This is a predicate over Y
- Evals to true if the person Y is friends with everyone

Existential Quantification

Asserts that a predicate can be satisfied

Example:

- $\exists(X)(Raining(X) \wedge \text{not } Cloudy(X))$
- Asserts that it is possible for it to rain when it is not cloudy

Example:

- $\text{not } \exists(X, Y)(Friends(X, Y) \wedge Friends(X, Z) \wedge Friends(Y, Z))$
- This is a predicate over Z
- Evals to true if Z isn't friends with two people who are also friends

Important Equivalence

$\forall(X)(P(X))$ is equivalent to...

not $\exists(X)(\text{not } P(X))$

- Ex: not $\exists(X,Y)(Friends(X,Y) \wedge Friends(X,Z) \wedge Friends(Y,Z))$
  is equivalent to
  $\forall(X,Y)(Friends(X,Z) \wedge Friends(Y,Z) \rightarrow \text{not } Friends(X,Y))$

Why important?

- Often easier to reason about $\exists$ compared to $\forall$
- Can be hard to wrap brain around an assertion that something is true over every item in the entire universe!
- In fact, SQL does not even have $\forall$

Questions?